UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

Proba scrisă a examenului de licenţă
Informatică Română

VARIANTA 1

NOTĂ

- Toate subiectele sunt obligatorii. La toate subiectele se cer rezolvări cu soluții complete.
- Nota minimă ce asigură promovarea este 5.00.
- Timpul efectiv de lucru este de 3 ore.

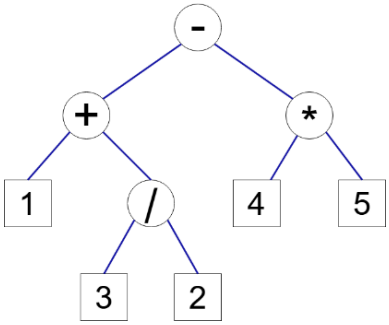
SUBIECT Algoritmică și programare

- **Se va indica limbajul de programare folosit.**
- **În implementările cerute nu este necesară dealocarea zonelor de memorie alocate dinamic.**
- **Lipsa unui stil de programare adecvat (denumiri sugestive pentru variabile, indentarea codului, comentarii dacă sunt necesare, lizibilitatea codului) duce la pierderea a 10% din punctajul aferent subiectului.**
- **Nu adăugați alte atribute, metode, în afara celor menționate în enunț, cu excepția constructorilor și a destructorilor, dacă e cazul. Nu modificați vizibilitatea atributelor specificate în enunț.**
- **Nu se vor folosi containere sortate, operații de sortare sau căutare predefinite.**
- **Pentru tipurile de date puteți folosi biblioteci existente (Python, C++, Java, C#).**

O expresie aritmetică se poate reprezenta sub forma unui arbore binar în felul următor. Dacă expresia este formată dintr-un singur operand, arborele binar asociat are un singur nod (nodul rădăcină), care conţine ca informaţie utilă operandul respectiv. Dacă expresia este de forma **E1** *op* **E2**, unde *op* este un operator binar iar **E1** şi **E2** sunt expresii aritmetice, acesteia i se asociază un arbore binar care are nodul rădăcină etichetat cu operatorul *op*, subarborele stâng este arborele binar asociat expresiei **E1**, iar subarborele drept este arborele binar asociat expresiei **E2**.

Prin parcurgerea în *postordine* a arborelui binar asociat unei expresii aritmetice se obține *forma postfixată* a acesteia. ***De exemplu***, pentru expresia (1+3/2)-4*5

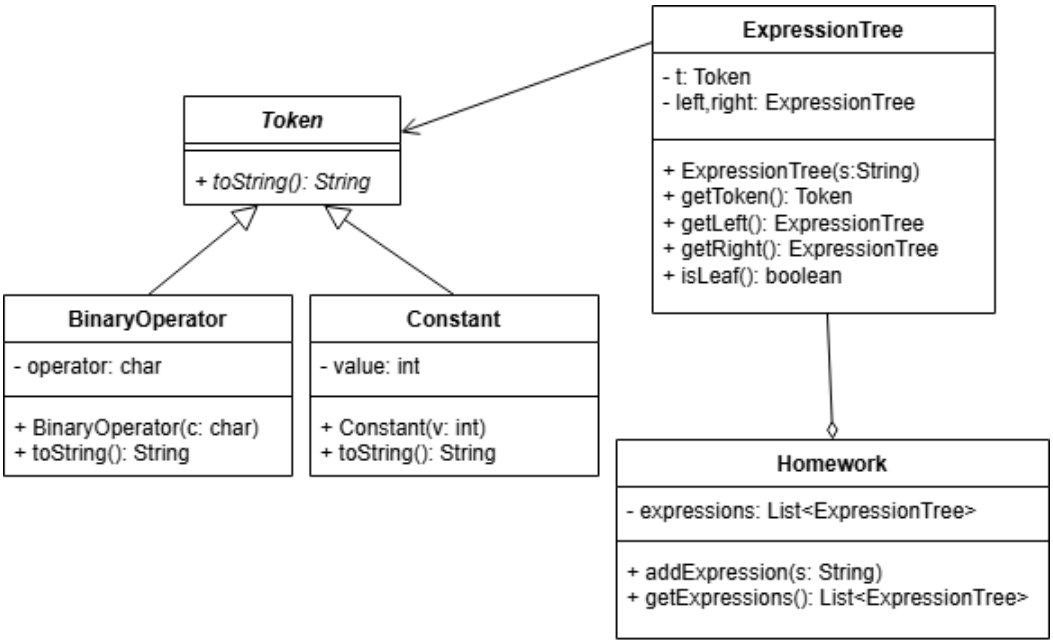
- arborele binar asociat este



- forma *postfixată* este '132/+45*-'.

1. (**3.5 puncte**) Scrieți o funcție într-unul din limbajele de programare **Python, C++, Java** sau **C#**, care primește ca parametru un șir *s* de cel mult 100 caractere reprezentând forma *postfixată* **corectă** a unei expresii aritmetice care conține operatorii binari '+' (adunare), '-' (scădere), '*' (înmulțire), '/' (împărțire), iar ca operanzi numere naturale nenule de o cifră. Funcția va returna valoarea expresiei aritmetice. Algoritmul trebuie să aibă complexitatea timp $\theta(n)$, unde *n* este lungimea șirului de caractere *s*. Soluțiile care nu se încadrează în complexitatea cerută se punctează parțial. **Exemplu:** pentru șirul '132/+45*-' se va returna valoarea **-17.5**. Se pot implementa funcții auxiliare.
2. (**1 punct**) Precizați complexitatea timp în cazurile *favorabil*, *mediu* și *defavorabil* pentru funcția de la punctul **1.** Justificați răspunsul.

3. (**4.5 puncte**) Se dă următoarea diagramă UML conținând clasele **ExpressionTree** (reprezintă arborele binar asociat unei expresii aritmetice corecte), ***Token*** (Simbol), **BinaryOperator** (OperatorBinar), **Constant** (Constantă) și **Homework** (TemăDeCasă).



- Clasa **ExpressionTree** reprezintă arborele binar asociat unei expresii aritmetice ai cărei operanzi sunt constante (numere naturale nenule), iar operatorii sunt binari (+, -, *, /).
  - Constructorul clasei **ExpressionTree** primește ca parametru un șir de caractere nevid *s* reprezentând o expresie aritmetică corectă conținând conținând paranteze, operatori binari și constante ca operanzi, și construiește arborele binar asociat acesteia. *Acest constructor nu se cere a fi implementat*.
  - Metoda **isLeaf**() returnează adevărat dacă arborele este format dintr-un singur nod (este asociat unei expresii aritmetice care conține un singur operand) și fals în caz contrar.
  - Metoda **getToken**() returnează informația utilă stocată în rădăcina arborelui binar **A** asociat unei expresii nevide, iar metodele **getLeft**() și **getRight**() returnează subarborele stâng, respectiv subarborele drept al arborelui **A**.
- Clasa abstractă ***Token*** reprezintă informația utilă (operator binar sau operand constant) care poate fi stocată într-un nod al arborelui. Clasa are o metodă abstractă ***toString***() care returnează un șir de caractere asociat simbolului: în cazul în care simbolul este operator binar se returnează caracterul care reprezintă operatorul (ex: '+'); în cazul în care simbolul este o constantă se returnează valoarea constantei transformată în șir de caractere (de exemplu, '123').
- Clasa **Homework** memorează o listă *expressions* de expresii aritmetice primite ca temă de casă de către un elev. Metoda **addExpression(s*:* String)** din clasa **Homework** primește ca parametru un șir de caractere *s* reprezentând o expresie aritmetică corectă și adaugă arborele binar construit din *s* la finalul listei *expressions* . Metoda **getExpressions()** returnează lista de arbori binari asociați expresiilor aritmetice primite ca temă.

Scrieți un program într-unul din limbajele de programare **C++, Java**, sau **C#**, cu următoarele cerințe:

a) Declarați toate clasele, atributele și metodele conform diagramei de mai sus. Implementați doar următoarele metode:

a1) Constructorul clasei **BinaryOperator**. Atributul *operator* trebuie să reprezinte un operator aritmetic binar (+, -, *, /). Constructorul trebuie să impună constrângerea.
a2) Metoda **isLeaf**() din clasa **ExpressionTree**.
a3) Metoda **addExpression**(*s*: **String**) din clasa **Homework.**

b) Definiți o funcție care primește ca parametru un obiect *e* de tip **ExpressionTree** și returnează șirul de caractere reprezentând forma postfixată a expresiei aritmetice asociate arborelui memorat în *e*.

c) Construiți un obiect *h* de tip **Homework** în care se adaugă următoarele expresii aritmetice de evaluat: '(5-3)+(2*6)', '5-(3*4+5)/(6-2)' și '3*(5/(3+2)-4)+6'. Pentru fiecare expresie aritmetică memorată în *h* afișați forma postfixată a acesteia, folosind funcția de la **b)**.

SUBIECT Baze de date **VARIANTA 1**

Problema 1. (4 puncte)

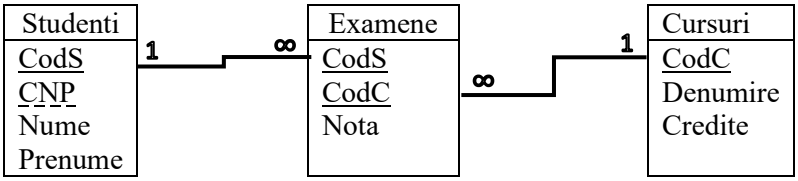
O firmă care produce prosoape stochează informații despre clienți, prosoape și recenzii ale clienților legate de prosoape într-o bază de date relațională:

- Un client are un cod, un nume, o adresă de email (unică pentru fiecare client) și data nașterii.
- Un prosop are un cod, o denumire, o descriere, un model, lungime și lățime.
- Un material are un cod, un nume și o descriere. Un material poate fi folosit la fabricarea mai multor prosoape, iar un prosop poate fi fabricat din mai multe materiale. Fiecare material poate fi asociat fiecărui prosop cel mult o singură dată. Pentru fiecare pereche formată dintr-un prosop și un material (de exemplu perechea formată din prosopul p1 și materialul m1) se va stoca și valoarea procentuală a materialului din compoziția prosopului.
- Un client poate da note mai multor prosoape, iar un prosop poate primi note de la mai mulți clienți. Fiecare client poate da fiecărui prosop cel mult o notă.

Realizați o schemă relațională BCNF pentru baza de date, evidențiind riguros cheile primare, cheile candidat și cheile externe. Realizați schema într-una din manierele indicate în exemplul de mai jos:

* exemplu pentru tabelele Studenti, Cursuri și Examene:

V1. Diagramă cu tabele, chei primare subliniate cu linie continuă, chei candidat subliniate cu linie întreruptă, legături trasate direct între cheile externe și cheile primare / candidat corespunzătoare (de exemplu, legătură trasată între coloana CodS din Examene și coloana CodS din Studenti).



V2.
Studenti[CodS, CNP, Nume, Prenume]
Cursuri[CodC, Denumire, Credite]
Examene[CodS, CodC, Nota]
Cheile primare sunt subliniate cu linie continuă, iar cheile candidat sunt subliniate cu linie întreruptă. {CodS} este cheie externă în Examene și face referire la {CodS} din Studenti. {CodC} este cheie externă în Examene și face referire la {CodC} din Cursuri.

Problema 2. (5 puncte)

Considerăm următoarele relații dintr-o bază de date despre o firmă care oferă spre închiriere jocuri de societate:

Categorii(CodCategorie, Denumire)
JocuriDeSocietate(CodJocDeSocietate, Denumire, *CodCategorie*, AnAparitie, NrMinJucatori, NrMaxJucatori, PretPeZi)
Persoane(CodPersoana, Nume, NrTelefon)
Inchirieri(CodInchiriere, *CodJocDeSocietate*, *CodPersoana*, DataInchirierii, NrZile, SumaPlatita)

Cheile primare sunt subliniate. Cheile externe sunt scrise cursiv și au aceeași denumire cu coloanele la care fac referire.

a. Scrieți o interogare SQL care returnează codul și denumirea jocurilor de societate care pot fi jucate individual (vezi atributul NrMinJucatori), au fost închiriate în anul 2024 cel puțin o dată și au anul de lansare egal cu anul cel mai recent al tuturor lansărilor de jocuri de societate care îndeplinesc condițiile menționate (vezi atributul AnAparitie).

b. Se dau instanțele următoare ale relațiilor JocuriDeSocietate, Categorii, Persoane și Inchirieri:

Persoane:

Cod Persoana	Nume	NrTelefon
1	P1	1111111111
2	P2	2222222222
3	P3	3333333333

Categorii:

Cod Categorie	Denumire
1	joc de strategie
2	joc de familie
3	joc cooperare

JocuriDeSocietate:

CodJocDe Societate	Denumire	Cod Categorie	An Aparitie	NrMin Jucatori	NrMax Jucatori	Pret PeZi
1	JS1	1	2020	3	6	15
2	JS2	2	2016	2	4	5
3	JS3	1	2020	2	5	12
4	JS4	3	2024	3	10	10

Inchirieri:

Cod Inchiriere	CodJocDe Societate	Cod Persoana	Data Inchirierii	NrZile	Suma Platita
1	1	1	15.05.2025	3	45
2	2	1	18.05.2025	5	25
3	3	2	10.05.2025	2	24
4	3	2	02.06.2025	3	36
5	1	3	25.05.2025	1	15
6	2	3	03.06.2025	4	20
7	3	3	10.06.2025	2	24
8	4	2	20.06.2025	3	30

b1. Precizați rezultatul evaluării interogării de mai jos pe instanțele date. Menționați strict valorile tuplului / tuplurilor și denumirile coloanelor din rezultat fără a prezenta toți pașii evaluării interogării.

```
SELECT P.CodPersoana, P.Nume, MIN(SumaPlatita) MinSum
FROM Persoane P
    INNER JOIN Inchirieri I ON P.CodPersoana = I.CodPersoana
    INNER JOIN JocuriDeSocietate JS ON I.CodJocDeSocietate = JS.CodJocDeSocietate
    INNER JOIN Categorii C ON JS.CodCategorie = C.CodCategorie
WHERE C.Denumire = 'joc de strategie'
GROUP BY P.CodPersoana, P.Nume
HAVING COUNT(DISTINCT I.CodJocDeSocietate) =
    (SELECT COUNT(*)
     FROM JocuriDeSocietate JS2
          INNER JOIN Categorii C2 ON JS2.CodCategorie = C2.CodCategorie
     WHERE C2.Denumire = 'joc de strategie'
     )
```

b2. Explicați dacă următoarele dependențe funcționale sunt satisfăcute sau nu de datele din instanța Inchirieri:

- {CodInchiriere} → {CodJocDeSocietate}
- {CodPersoana} → {DataInchirierii}.

VARIANTA 1

SUBIECT Sisteme de operare

Problema 1 (5 puncte). Răspundeţi la următoarele întrebări despre execuţia programului de mai jos, considerând că toate include-urile necesare sunt prezente.

```
 1  int main() {
 2      int k;
 3      if(fork() < 0) {
 4          printf("abc\n");
 5      } else if(fork() == 0) {
 6          return 1;
 7      } else {
 8          return 2;
 9      }
10      while((k = wait(NULL)) > 0) {
11          printf("%d\n", k);
12      }
13      return 3;
14  }
```

a) Explicaţi în detaliu funcţionarea liniei 3

b) Explicaţi în detaliu funcţionarea liniilor 10-12

c) Ce se va afişa în consolă dacă ambele apeluri `fork` eşuează? Justificaţi răspunsul.

d) Ce se va afişa în consolă dacă ambele apeluri `fork` se execută cu succes? Justificaţi răspunsul.

e) Ce va afişa în consolă procesul iniţial dacă se elimină linia 8 şi ambele apeluri `fork` se execută cu succes? Justificaţi răspunsul.

Problema 2 (4 puncte). Răspundeţi la următoarele întrebări despre execuţia scriptului `./a.sh` de mai jos.

```
1   #!/bin/bash
2
3   N=$1
4   D=$2
5   shift 2
6
7   for A in $*; do
8       if [ $N -gt 0 ]; then
9           mkdir -p $D/$A
10          $0 `expr $N - 1` $D/$A $*
11      else
12          echo abc > $D/$A
13      fi
14  done
```

a) Explicaţi în detaliu funcţionarea liniei 10.

b) Ce valori va lua variabila `A` la rularea comenzii de mai jos?

```
./a.sh 1 a b c d
```

c) Considerând că directorul `x` nu există iniţial, ce se va afişa în consolă după rularea comenzilor de mai jos?

```
./a.sh 1 x a b
find x -type f | sort
```

d) Considerând că directorul `y` nu există iniţial, ce se va afişa în consolă după rularea comenzilor de mai jos?

```
./a.sh 2 y a b
cat `find y -type f` | wc -l
```

BAREM INFORMATICĂ
VARIANTA 1

Subiect Algoritmică și Programare

Oficiu – **1p**

Cerința **1**. **– 3.5p**
- signatura – **0.2p**
- implementare având complexitate timp $\theta(n)$ – **3.2p**
  * soluție având complexitate timp $O(n^2)$ – **2**p
- returnare rezultat – **0.1p**

Cerința **2**. **– 1p**
- caz favorabil **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz mediu **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz defavorabil **0.2p** din care
  complexitate – 0.1
  justificare – 0.1

Cerința **3.a) – 2.25p**

Definirea clasei Token – **0.2p** din care
- clasă abstractă – 0.1
- metoda **toString** – 0.1

Definirea clasei BinaryOperator– **0.65p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- **constructor (a1)** – 0.3
- metoda **toString** – 0.1

Definirea clasei Constant– **0.45p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- constructor – 0.1
- metoda **toString** – 0.1

Definirea clasei ExpressionTree – **0.55p** din care
- atribute– 0.1
- constructor – 0.1
- metode **getToken, getLeft, getRight** – 0.15
- metoda **isLeaf (a2) –** 0.2

Definirea clasei Homework – **0.4p** din care
- atribut – 0.1
- metoda **getExpressions** – 0.1p
- metoda **addExpression (a3)** – 0.2p

Funcția **3.b)** – **1.75p**
- signatura – **0.1p**
- implementare parcurgere postordine – **1.55p**
- returnare rezultat – **0.1p**

Funcția principală **3.c)** – **0.5p**
- construire obiect **h** – 0.05p
- adăugare expresii aritmetice în **h** - 0.25p
- parcurgere listă expresii memorate în **h** și afișare formă postfixată – 0.2p din care
  apel funcție b) pentru expresie– 0.1p
  afișare formă postfixată expresie - 0.1 p

BAREM INFORMATICĂ
VARIANTA 1

Subiect Baze de date

Oficiu – **1p**

Problema 1. Punctaj - 4p

- relații cu atribute corecte, chei primare, chei candidat: **3p**
- legături modelate corect (chei externe): **1p**

Problema 2. Punctaj - 5p

- **a** - rezolvarea completă a interogării: **2.5p**

- **b1** - rezultat evaluare interogare:

CodPersoana	Nume	MinSum
3	P3	15

  - coloane – **0.5p**
  - valori tuplu – **1p**

- **b2 -** {CodInchiriere} → {CodJocDeSocietate} este satisfăcută – **0.25p**; **0.25p** explicație
  - {CodPersoana} → {DataInchirierii} nu este satisfăcută – **0.25p**; **0.25p** explicație

Notă: La specializările Informatică engleză și Informatică maghiară se iau în considerare versiunile traduse în limbile corespunzătoare.

VARIANTA 1

SUBIECT Sisteme de operare

Barem:

1p – oficiu

Problema 1 (5p)

1p – a) Se rulează `fork` şi dacă eşuează condiţia va fi adevărată. Dacă `fork` nu eşuează, se creează un proces fiu şi condiţia va fi evaluată ca falsă şi de procesul părinte şi de procesul fiu.

1p – b) Câtă vreme `wait` returnează succes, se tipăreşte PID-ul procesului fiu care tocmai s-a încheiat.

1p – c) Se va afişa `abc` pentru că procesul iniţial va intra în primul if, dar nu va intra în `while` pentru că neavând procese fiu, `wait` va returna eroare.

1p – d) Nu se va afişa nimic, pentru că procesul iniţial şi primul proces fiu se încheie la linia 8, al doilea proces fiu şi procesul nepot se incheie la linia 6

1p – e) Afişează PID-urile celor două procese fiu pe care le creează la liniile 3 şi 5.

Problema 2 (4p)

1p – a) Scriptul Shell se relansează în execuţie, cu primul argument decrementat, al doilea argument directorul tocmai creat şi restul argumentelor identice cu cele rămase după shift.

1p – b) Va lua valorile `b`, `c` şi `d`

1p – c) Se va afişa: `x/a/a  x/a/b  x/b/a  x/b/b`

1p – d) Se va afişa cifra 8